INGENIERÍA DEL SOFTWARE II

PROYECTO- PATRONES DE DISEÑO

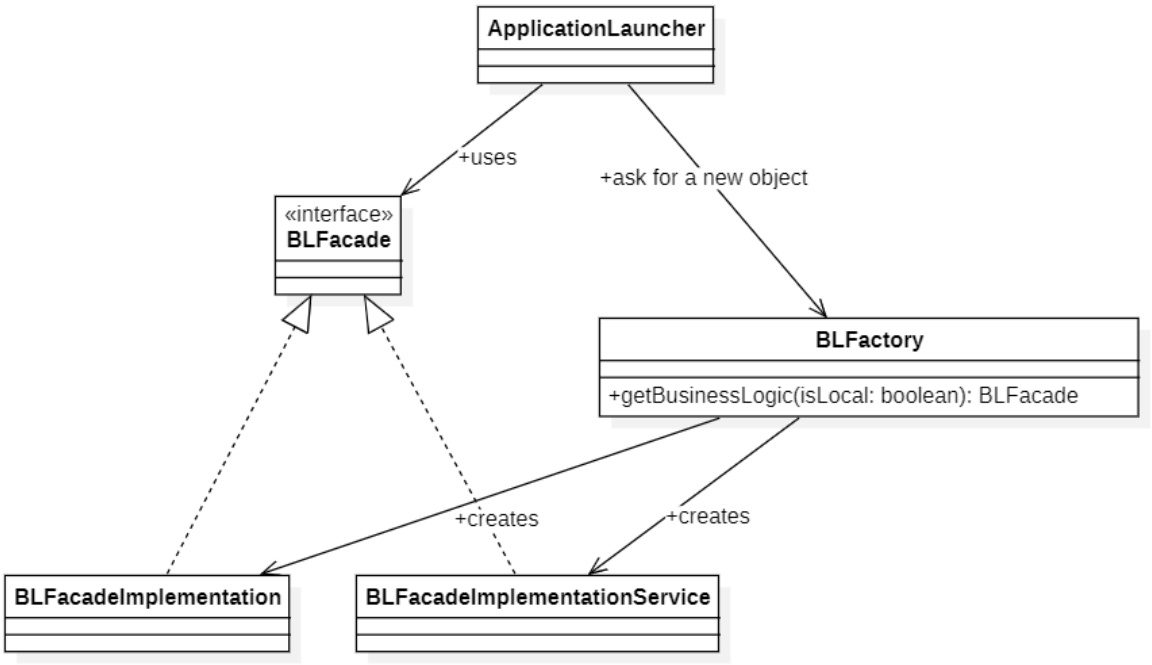
2025 - 2026

AUTORES:
Ilargi Quetzalli Matilde López
Gontzal Leizabe Escartin

ÍNDICE

1. PATRÓN FACTORY METHOD

Diagrama UML



Cambios realizados

Antes de aplicar el patrón Factory Method:
La clase *ApplicationLauncher* tenía un bloque *if/else* que se encargaba de decidir qué implementación de *BLFacade* se debía crear.
Si la lógica era local, se instanciaba un *BLFacadeImplementation* con un nuevo *DataAccess*.
Si la lógica era remota, se creaba un servicio web mediante *service.getPort(BLFacade.class)*.

Después de aplicar el patrón Factory Method:
Se ha creado una nueva clase *BLFactory* cuya responsabilidad es la creación de objetos de tipo *BLFacade*.
Dentro de esta clase se ha definido un nuevo método *getBusinessLogic(boolean isLocal)*.
Además, ahora la clase *ApplicationLauncher* ya no cuenta con el bloque *if/else* del principio, sino que invoca a la factoría de la siguiente forma: *appFacadeInterface = new BLFactory().getBusinessLogic(isLocal);* .

Finalidad de cada clase/interfaz

ApplicationLauncher
Su papel es: *Client*.
Su finalidad es invocar a la factoría *BLFactory* para obtener un objeto de tipo *BLFacade*, sin tener que conocer qué implementación se está utilizando.

BLFacade
Su papel es: *Product*.
Su finalidad es actuar como la interfaz común que comparten todas las implementaciones de la lógica de negocio creadas por la factoría. Gracias a *BLFacade*, *BLFactory* puede devolver diferentes implementaciones en el caso de que la ejecución de la aplicación sea local o remoto, sin tener que alterar las demás clases.

BLFacadeImplementation
Su papel es: *ConcreteProduct*.
Su finalidad es proporcionar la lógica de negocio cuando la aplicación se ejecuta localmente.

BLFacadeImplementationService
Su papel es: *ConcreteProduct*.
Su finalidad es proporcionar la lógica de negocio cuando la aplicación se ejecuta remotamente. Permite obtener la implementación de *BLFacade* mediante un servicio web.
En el diagrama UML se representa como una clase para mostrar la implementación remota, la aplicación la genera automáticamente mediante la llamada *service.getPort(BLFacade.class)* en la clase *BLFactory*.

BLFactory
Su papel es: *Creator*.
Su finalidad es decidir qué implementación devolver en caso de que la aplicación se ejecute localmente o remotamente. Además, evita que la lógica de presentación conozca las clases concretas.

Código modificado

ANTES

AplicationLauncher

```
package gui;

import java.net.URL;
import java.util.Locale;
```

```
import javax.swing.UIManager;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import configuration.ConfigXML;
import dataAccess.DataAccess;
import businessLogic.BLFacade;
import businessLogic.BLFacadeImplementation;

public class ApplicationLauncher {

   public static void main(String[] args) {

      ConfigXML c = ConfigXML.getInstance();

      System.out.println(c.getLocale());

      Locale.setDefault(new Locale(c.getLocale()));

      System.out.println("Locale: " + Locale.getDefault());

      try {

         BLFacade appFacadeInterface;
         UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

         if (c.isBusinessLogicLocal()) {

            DataAccess da = new DataAccess();
            appFacadeInterface = new BLFacadeImplementation(da);

         }

         else { // If remote

            String serviceName = "http://" + c.getBusinessLogicNode() + ":" +
c.getBusinessLogicPort() + "/ws/"
                  + c.getBusinessLogicName() + "?wsdl";

            URL url = new URL(serviceName);

            // 1st argument refers to wsdl document above
            // 2nd argument is service name, refer to wsdl document above
            QName qname = new QName("http://businessLogic/",
"BLFacadeImplementationService");

            Service service = Service.create(url, qname);

            appFacadeInterface = service.getPort(BLFacade.class);
         }

         MainGUI.setBussinessLogic(appFacadeInterface);
         MainGUI a = new MainGUI();
```

```
            a.setVisible(true);

        } catch (Exception e) {
            // a.jLabelSelectOption.setText("Error: "+e.toString());
            // a.jLabelSelectOption.setForeground(Color.RED);

            System.out.println("Error in ApplicationLauncher: " + e.toString());
        }
        // a.pack();

    }

      }
```

AHORA

ApplicationLauncher

```
package gui;

import java.util.Locale;

import javax.swing.UIManager;

import configuration.ConfigXML;
import businessLogic.BLFacade;
import businessLogic.BLFactory;

public class ApplicationLauncher {

    public static void main(String[] args) {

        ConfigXML c = ConfigXML.getInstance();

        System.out.println(c.getLocale());

        Locale.setDefault(new Locale(c.getLocale()));

        System.out.println("Locale: " + Locale.getDefault());

        try {

            BLFacade appFacadeInterface;
            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

            boolean isLocal = c.isBusinessLogicLocal();
            appFacadeInterface = new BLFactory().getBusinessLogic(isLocal);
```

```
            MainGUI.setBussinessLogic(appFacadeInterface);
            MainGUI a = new MainGUI();
            a.setVisible(true);

        } catch (Exception e) {
            // a.jLabelSelectOption.setText("Error: "+e.toString());
            // a.jLabelSelectOption.setForeground(Color.RED);

            System.out.println("Error in ApplicationLauncher: " + e.toString());
        }
        // a.pack();

    }

}
```

BLFactory

```
package businessLogic;

import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import configuration.ConfigXML;
import dataAccess.DataAccess;

public class BLFactory {

    public BLFacade getBusinessLogic(boolean isLocal) throws MalformedURLException {
        if (isLocal) {

            DataAccess da = new DataAccess();
            return new BLFacadeImplementation(da);

        }

        else { // If remote

            ConfigXML c = ConfigXML.getInstance();

            String serviceName = "http://" + c.getBusinessLogicNode() + ":" +
c.getBusinessLogicPort() + "/ws/"
                    + c.getBusinessLogicName() + "?wsdl";

            URL url = new URL(serviceName);
```

```
            // 1st argument refers to wsdl document above
            // 2nd argument is service name, refer to wsdl document above
            QName qname = new QName("http://businessLogic/",
"BLFacadeImplementationService");

            Service service = Service.create(url, qname);

            return service.getPort(BLFacade.class);
        }
    }
}
```

Por un lado en la clase *ApplicationLauncher* se ha sustituido el bloque *if/else* por una llamada a la factoría *BLFactory* de la siguiente forma:

```
boolean isLocal = c.isBusinessLogicLocal();
appFacadeInterface = new BLFactory().getBusinessLogic(isLocal);
```

Con este cambio, la creación del objeto de tipo *BLFacade* deja de hacerse en la capa de presentación y pasa a delegarse en la factoría.

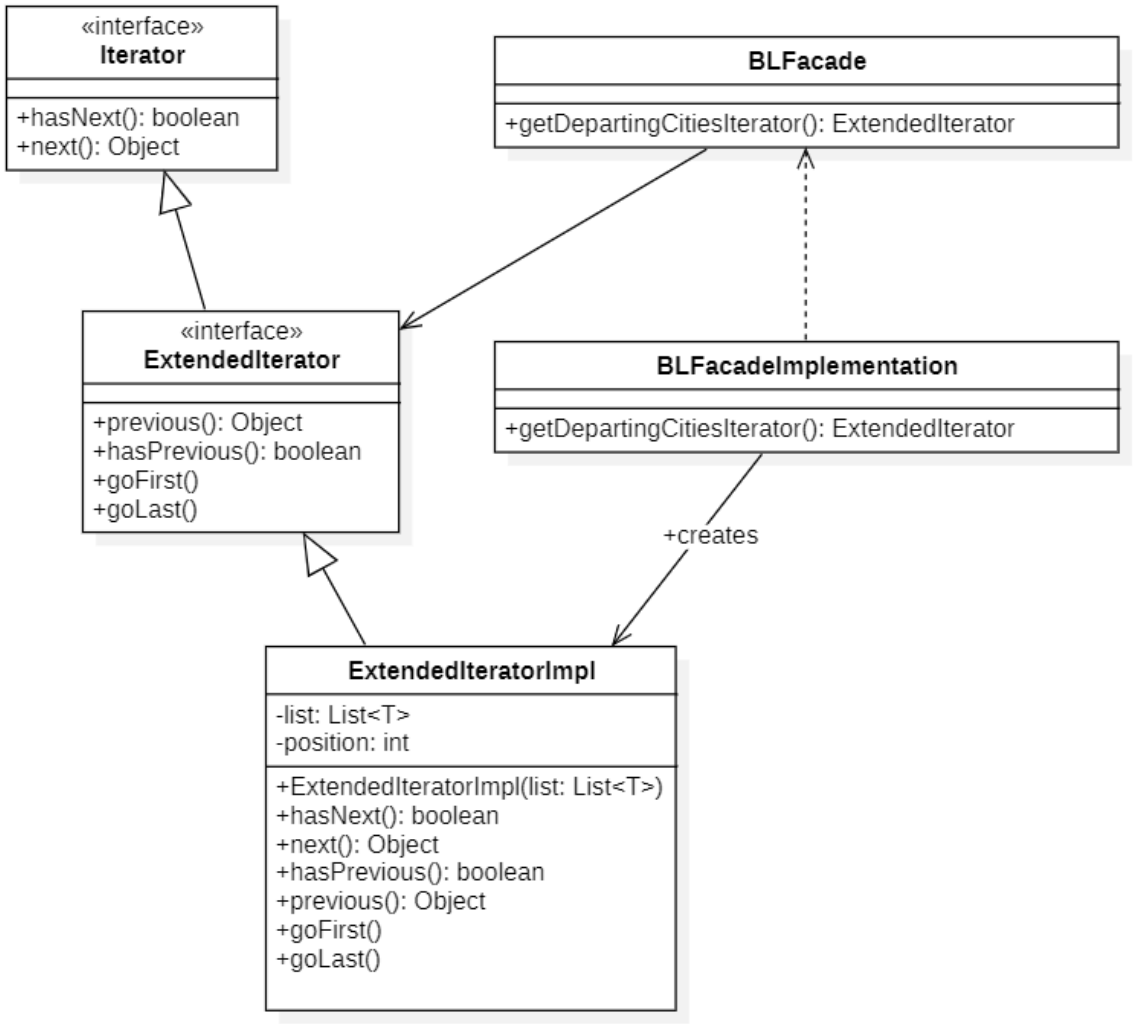
Por otro lado, se ha creado una nueva clase *BLFactory* que contiene el siguiente método:

```
public BLFacade getBusinessLogic(boolean isLocal)
```

Este método encapsula la lógica de selección del servicio. La factoría asume la responsabilidad que tenía anteriormente la clase *ApplicationLauncher*.
Si la aplicación se ejecuta localmente, devuelve un objeto de tipo *BLFacadeImplementation*.
Si se ejecuta de forma remota, obtiene la implementación mediante el servicio web con la llamada *service.getPort(BLFacade.class)*.

2. PATRÓN ITERATOR

Diagrama UML



Cambios realizados

He creado un nuevo paquete llamado Iterator donde he definido la interfaz ExtendedIterator, y las clases ExtendedIteratorImpl e IteratorTest. Adicionalmente he creado la función getDepartingCitiesIterator que devuelve un objeto iterator para el arrayList de DepartCities.

Finalidad de cada clase/interfaz

En Extendediterator estan definidos los métodos a implementar, en ExtendedIteratorImpl esta hecha la implementación y en IteratorTest he hecho las pruebas para verificar el correcto funcionamiento de la clase.

Código modificado

En BLFacade se ha añadido la siguiente función:

```
public ExtendedIterator<String> getDepartingCitiesIterator();
```

En BLFacadeImplementation se ha añadido la siguiente función:

```
public ExtendedIterator<String> getDepartingCitiesIterator() {
   List<String> cities = getDepartCities();
   return new ExtendedIteratorImpl<String>(cities);
}
```

Se ha creado ExtendedIterator:

```
package Iterator;

import java.util.Iterator;

public interface ExtendedIterator<Object> extends Iterator<Object> {

   public Object previous();

   public boolean hasPrevious();

   public void goFirst();

   public void goLast();
}
```

Se ha creado ExtendedIteratorImpl:

```
package Iterator;

import java.util.List;

public class ExtendedIteratorImpl<T> implements ExtendedIterator<T> {

   private List<T> list;
```

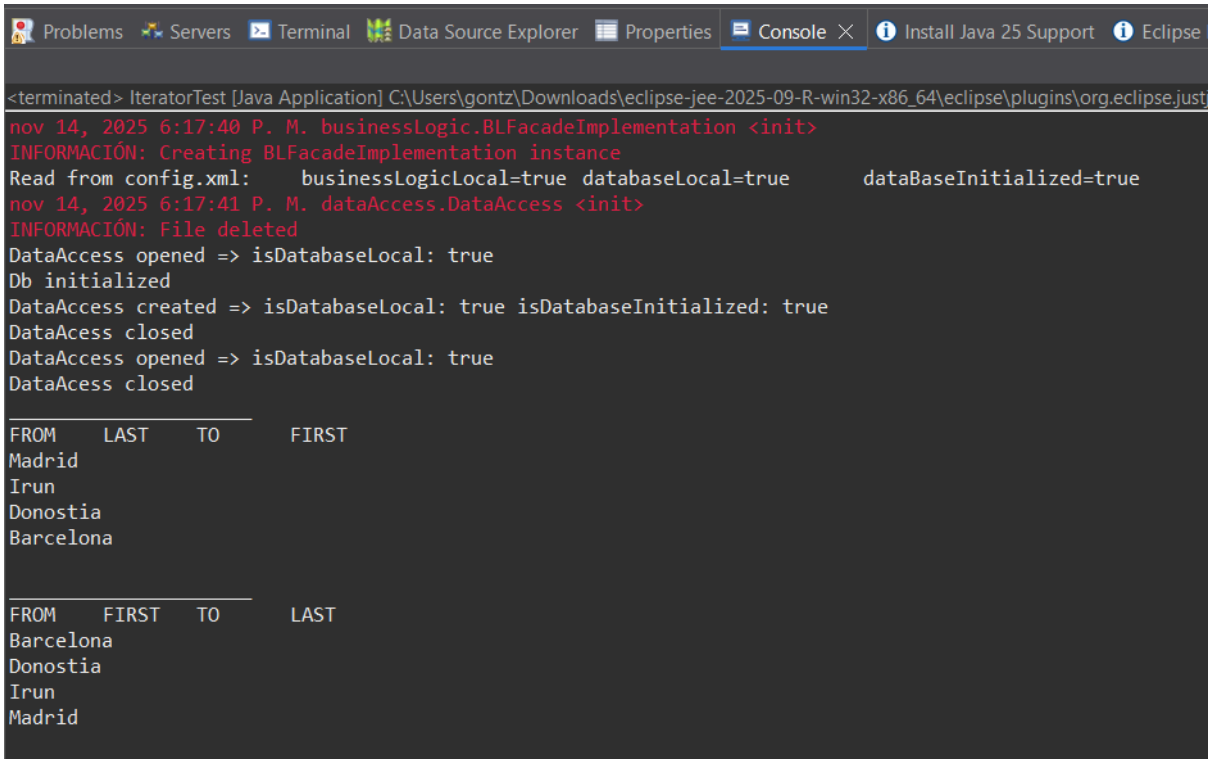
```
    private int position;

    public ExtendedIteratorImpl(List<T> list) {
        this.list = list;
        this.position = 0; // al inicio
    }

    @Override
    public boolean hasNext() {
        return position < list.size();
    }

    @Override
    public T next() {
        return list.get(position++);
    }


    @Override
    public boolean hasPrevious() {
        return position > 0;
    }

    @Override
    public T previous() {
        position--;
        return list.get(position);
    }

    @Override
    public void goFirst() {
        position = 0;
    }

    @Override
    public void goLast() {
        position = list.size();
    }
}
```

Se ha creado IteratorTest:

```
package Iterator;
import businessLogic.*;

public class IteratorTest {
```

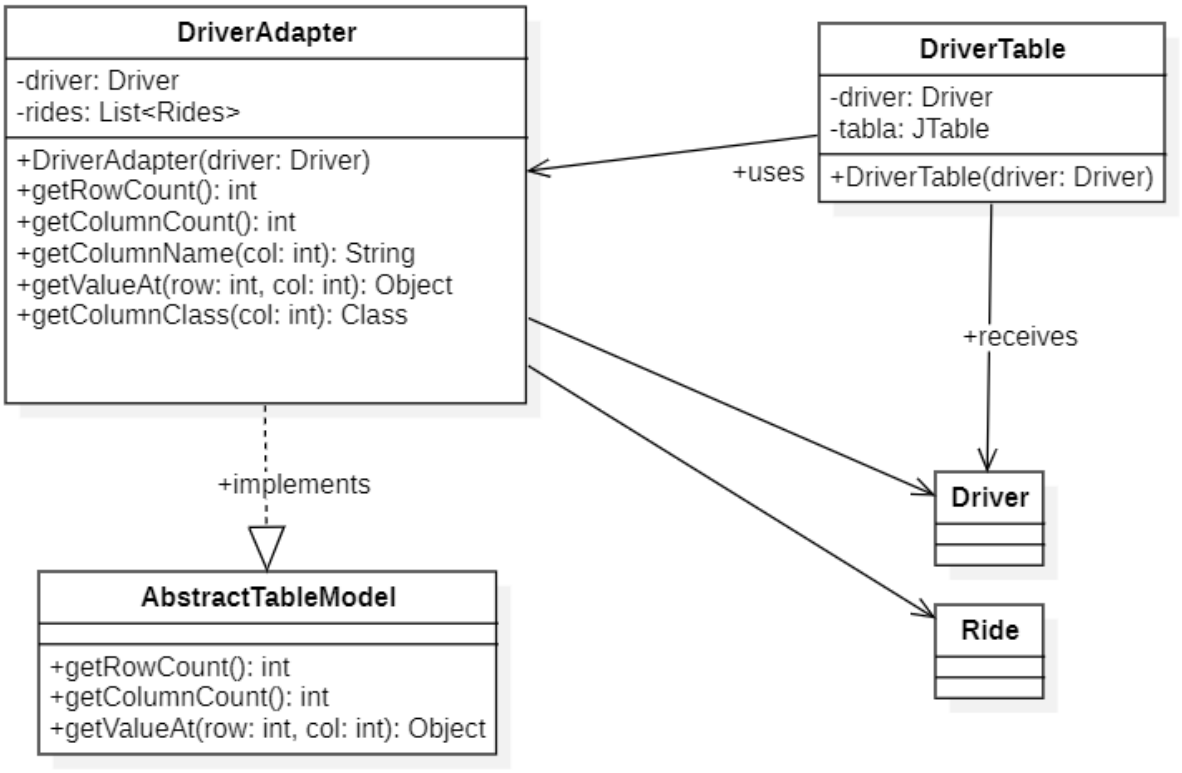
```
	public static void main(String[] args) {
//		the BL is local
	boolean isLocal = true;
	BLFacade blFacade = new BLFacadeImplementation();
	ExtendedIterator<String> i = blFacade.getDepartingCitiesIterator();
	String c;
	System.out.println("_____");
	System.out.println("FROM LAST TO FIRST");
	i.goLast(); // Go to last element
	while (i.hasPrevious()) {
	c = i.previous();
	System.out.println(c);
	}
	System.out.println();
	System.out.println("_____");
	System.out.println("FROM FIRST TO LAST");
	i.goFirst(); // Go to first element
	while (i.hasNext()) {
	c = i.next();
	System.out.println(c);
	}
	}

}
```

Captura de imagen de ejecución


Problems
Servers
Terminal
Data Source Explorer
Properties
Console
Install Java 25 Support
Eclipse
<terminated> IteratorTest [Java Application] C:\Users\gontz\Downloads\eclipse-jee-2025-09-R-win32-x86_64\eclipse\plugins\org.eclipse.just
nov 14, 2025 6:17:40 P. M. businessLogic.BLFacadeImplementation <init>
INFORMACIÓN: Creating BLFacadeImplementation instance
Read from config.xml: businessLogicLocal=true databaseLocal=true dataBaseInitialized=true
nov 14, 2025 6:17:41 P. M. dataAccess.DataAccess <init>
INFORMACIÓN: File deleted
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true
DataAcess closed
DataAccess opened => isDatabaseLocal: true
DataAcess closed
FROM LAST TO FIRST
Madrid
Irun
Donostia
Barcelona
FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid

3. PATRÓN ADAPTER

Diagrama UML



Cambios realizados

Se ha creado un nuevo paquete llamado adapter, en el cual se ha implementado la clase DriverAdapter, utilizada como modelo de tabla (TableModel) para visualizar los viajes de un conductor en componente JTable.
Además, se ha añadido la clase DriverTable, que construye una interfaz gráfica en la que se muestran los datos de dicho conductor.

También se ha creado la clase AdapterTest para verificar que las clases anteriores funcionan correctamente.

Finalidad de cada clase/interfaz

DriverAdapter

Actuar como adaptador en el patrón adapter, convirtiendo un objeto Driver en un formato compatible con un TableModel

DriverTable

Es la clase responsable de la interfaz gráfica, crea un JFrame, configura el JTable y utiliza la clase DriverAdapter como modelo de datos para presentar visualmente los viajes del conductor

AdapterTest

Prueba para comprobar el correcto funcionamiento de DriverAdapter y DriverTable.

Código modificado

DriverAdapter:

```
package adapter;

import javax.swing.table.AbstractTableModel;

import domain.Driver;
import domain.Ride;

import java.util.List;

public class DriverAdapter extends AbstractTableModel {

    private Driver driver;
    private List<Ride> rides;

    private final String[] columnNames = {
            "Origin", "Destination", "Date", "Places", "Price"
    };

    public DriverAdapter(Driver driver) {
        this.driver = driver;
        this.rides = driver.getCreatedRides();
    }

    @Override
    public int getRowCount() {
        return rides.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }
```

```
    @Override
    public String getColumnName(int col) {
        return columnNames[col];
    }

    @Override
    public Object getValueAt(int row, int col) {
        Ride r = rides.get(row);

        switch (col) {
            case 0: return r.getFrom();
            case 1: return r.getTo();
            case 2: return r.getDate();
            case 3: return r.getnPlaces();
            case 4: return r.getPrice();
            default: return null;
        }
    }

    @Override
    public Class<?> getColumnClass(int col) {
        switch (col) {
            case 2: return java.util.Date.class;
            case 3: return Integer.class;
            case 4: return Double.class;
            default: return String.class;
        }
    }
}
```

DriverTable:

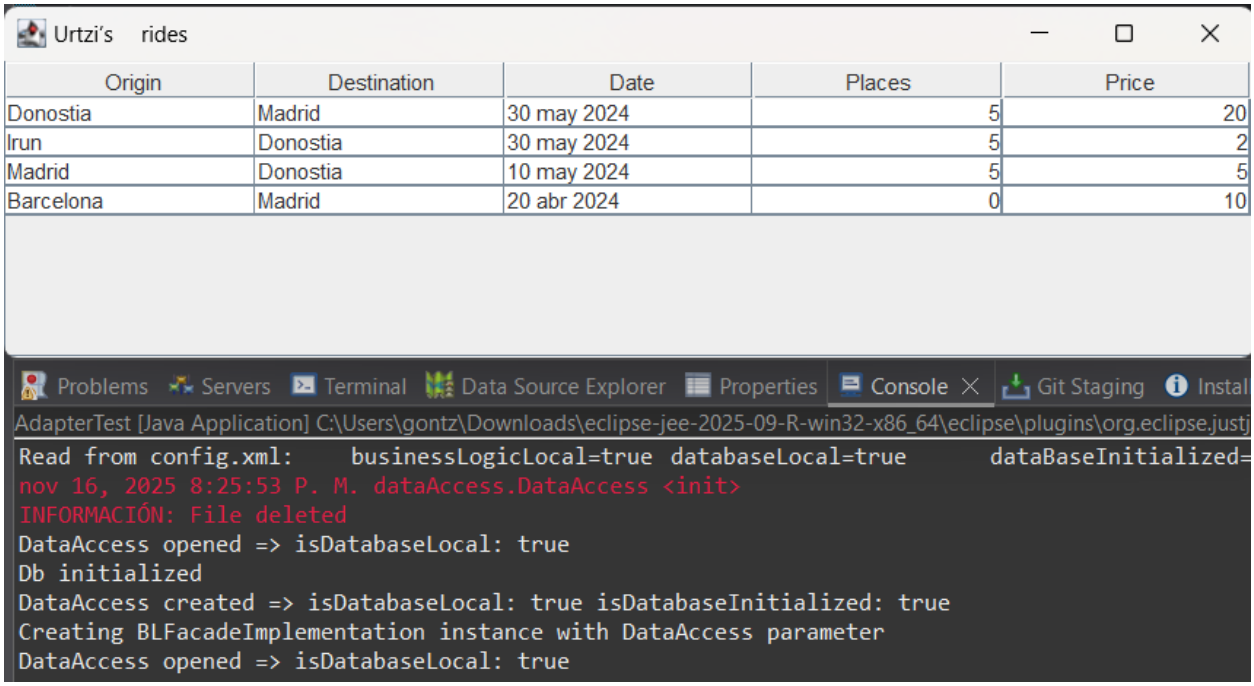
```
package adapter;

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import domain.Driver;

public class DriverTable extends JFrame {
        private Driver driver;
        private JTable tabla;
```

```
	public DriverTable(Driver driver) {
		super(driver.getUsername() + "'s	rides ");
		this.setBounds(100, 100, 700, 200);
		this.driver = driver;
		DriverAdapter adapt = new DriverAdapter(driver);
		tabla = new JTable(adapt);
		tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
//Creamos un JscrollPane	y	le agregamos la JTable
		JScrollPane scrollPane = new JScrollPane(tabla);
//Agregamos el	JScrollPane	al contenedor
		getContentPane().add(scrollPane, BorderLayout.CENTER);
	}
}
```

AdapterTest:

```
package adapter;

import businessLogic.BLFacadeImplementation;
import businessLogic.BLFactory;

import java.net.MalformedURLException;

import businessLogic.BLFacade;
import domain.Driver;

public class AdapterTest {
	public static void main(String[]	args) throws MalformedURLException	{
//		the	BL	is	local
	boolean isLocal =	true;
	BLFacade	blFacade =	new BLFactory().getBusinessLogic(isLocal);
	Driver	d= blFacade. getDriver("Urtzi");
	DriverTable	dt=newDriverTable(d);
	dt.setVisible(true);
	}
}
```

Captura de imagen de ejecución


Urtzi's rides
Origin
Destination
Date
Places
Price
Donostia
Madrid
30 may 2024
5
20
Irun
Donostia
30 may 2024
5
2
Madrid
Donostia
10 may 2024
5
5
Barcelona
Madrid
20 abr 2024
0
10
Problems
Servers
Terminal
Data Source Explorer
Properties
Console
Git Staging
AdapterTest [Java Application] C:\Users\gontz\Downloads\eclipse-jee-2025-09-R-win32-x86_64\eclipse\plugins\org.eclipse.justj
Read from config.xml: businessLogicLocal=true databaseLocal=true dataBaseInitialized=
nov 16, 2025 8:25:53 P. M. dataAccess.DataAccess <init>
INFORMACIÓN: File deleted
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true
Creating BLFacadeImplementation instance with DataAccess parameter
DataAccess opened => isDatabaseLocal: true